CEN/CLC/JTC 22/WG 3 "Quantum Computing and Simulation"

Convenor: **Paul Alexandra Mme**

ISA_requirements_V08

Document type	Related content	Document date	Expected action
Meeting / Document for discussion	Meeting: VIRTUAL 18 Jun 2025	2025-06-16	**COMMENT/REPLY** by 2025-06-18

Description

Dear members,

Please find attached the latest contribution to the Cryogenic Solid State document.

Kind regards,

CEN/CLC/JTC 22/WG 3 **N** ##

CEN/CLC/JTC 22/WG 3 "Quantum Computing and Simulation"
WG Secretariat: **xxNSBxx**
Convenor: **xxWGCHAIRxx**

CEN-CLC-JTC 22_##_Add_instruction_set_architecture_functionality

Document type	**Meeting**	**Document date**	**Expected action**
Contribution	JTC22-WG3-014	2025-06-06	For decision

Title	Proposal for adding instruction set architecture functionality
Authors	Juan Carlos Boschero (juan.boschero@tno.nl), Rob F.M. van den Brink (rob.vandenbrink@delft-circuits.com), Michele Amoretti (michele.amoretti@unipr.it) , Valentin Torggler (v.torggler@parityqc.com), Michael Fellner (m.fellner@parityqc.com)
Organisations	TNO, Delft Circuits, University of Parma, Parity QC
Representing	NEN, UNI, DIN, ASI
Work Item number	N/A
Work Item title	Cryogenic Solid-State Quantum Computing
Summary	This document proposes an addition to the JTC22 WG3 Cryogenic Solid-State Quantum Computing instruction set architecture functionality.
Motivation	There is a need to further define the descriptions and the information flows in the instruction set architecture.
Details (also next page)	See next pages. • Update chapter 8

The aim of this contribution is to identify all kinds of functional requirements that an ISA should at least support. This ISA is a functionality within the control software layer that can process instructions from higher layers, such as the HAL, and convert them into commands for control electronics to generate all kinds of pulses and measurements for qubits.
As such, this contribution gives lots of guidance on how to organize the instruction sets of ISA architectures.

8.2 Functional requirements

8.2.1 Instruction Set Architecture

The ISA is a functionality within the control software layer that can process instructions from higher layers, such as the HAL, and convert them into commands for control electronics to generate all kinds of pulses and measurements for qubits. It can process received instructions to initialize the setup, to inquire supported capabilities, to execute gates or measurements and to read-out intermediate results.
However, there are several limitations that should be accounted for:

- *Lane limitations:* Execution often means the firing of pulses by the control electronics to the qubits, preferably multiple pulses simultaneously to different qubits. However the available hardware limits the number of pulses that can be fired simultaneously. When more qubits are to be controlled than this limit, a switching matrix is to be used after the pulse generators to reach all qubits of interest. These switches can offer so called “lanes” to connect a qubit to a particular pulse generator. To save hardware such a switching matrix may support only a restricted combination of lanes, and higher layers such as the HAL should be aware of such limitations. A convenient way to specify the supported connections between pulse generators and qubits through a switching matrix is by means of a so called “lane matrix”.
- *Adjacency limitations*: Similar limitations do occur between qubits. Direct application of entangling operations between qubits can only be achieved between qubits that are adjacent to each other. It requires multiple steps to perform entangling operations between non-adjacent qubits, and therefore higher layers, such as the HAL, should be aware of such limitations. A convenient way to specify which qubits are edge-connected is by means of a so called “adjacency matrix”.

To offer all required functionalities, the ISA should support the following groups of instructions:

8.2.1.1. Initialization instructions

Initialization instructions are to prepare the setup for quantum computing tasks. These instructions are typically sent at the beginning of a calculation sequence by higher layers, such as the HAL. But they may also be re-sent as often as needed. At least the following instructions should be supported by the ISA:

- “*set shapes*”: construct a data structure with predefined wave shapes for pulses.
- “*set bases*”: construct a data structure with predefined bases to measure individual qubits.
- “*set mapping*”: construct qubit topological mapping.
- “*set registers*”: group the qubits into registers, and allocate indices to individual qubits. This grouping can be defined from higher layers, such as the HAL, or selected from one or more predefined register grouping.
- “*set lanes*”: allocate numbering to lanes of pulsing channels.
- “*run calibration*”: call a calibration from a set of predefined calibration procedures, to prepare a setup.

It might be possible that future systems will allow for a user-definable grouping of qubits to arbitrary registers. In that case the definitions of lane indices and mapping will also change.

8.2.1.2. Inquiring capability instructions

Inquiring capabilities is a mechanism for higher layers, such as a HAL, to identify relevant information about the setup, without performing any calculation. Most of these capabilities can be hard-coded in the software by the vendor, who has full knowledge about the hardware. But some of the capabilities can be a result stored in memory after performing initialization instructions.

As a remark, some systems can modify their registers during runtime. When the system has this capability, it should support instructions on how they can modify the register.

8.2.1.2.1 Inquiring the organization of qubits

Qubits can be grouped into “quantum registers” if they can be entangled, each with a unique index number to address the individual qubits. A system can support one or more of those registers, and if multiple registers are being used, each of them has a unique identifier to address the individual registers.

A quantum compiler or interpreter should have full knowledge of how the qubits are organized in order to optimize generated code. To prevent that each compiler is targeted only to a single system, it could start with inquiring these capabilities via the HAL, which in turn inquires the control software layer. The ISA should support at least the following capability instructions:

- *“Get register set”:* Return a list with identifiers of each of the supported quantum registers.
- *“Get register width”:* Return the number of available qubits for each individual quantum register.
- *“Get adjacency matrix”*: Returns an "adjacency matrix" for each individual quantum register, to specify which qubits in the register are edge-connected. For instance, when a register has N_q qubits, then this adjacency matrix C has size $N_q \times N_q$. The default of each element in this matrix is false, but if q_x and q_y are the indices of two adjacent qubits then $C(q_x,q_y)=C(q_y,q_x)=$true. Matrix C is therefore a symmetric matrix, since C(k,r)=C(r,k).
- “*Get lane matri*x”: Returns a "lane matrix" for each individual quantum register, to specify for each pulse generator to which qubit it can be connected. For instance, when N_p pulse generators can be connected to N_q qubits in that register, then the associated lane matrix L has size $N_p \times N_q$. The default of each element in this matrix is false, but if pulse generator p_x can be connected with qubit q_y then $L(p_x,q_y)=$true.
  Returning an empty matrix means that this limitation does not exist.

Different kind of qubits can be distinguished when inquiring system specifications, such as:

- **Physical qubit:** A noisy quantum system in which a two-dimensional Hilbert space can be encoded.
  - **Data qubit:** data qubits are physical qubits used for storing and processing quantum information.
  - **Measurement qubits:** physical qubits used for stabilizer measurements of quantum error-correcting codes.
  - **Communication qubit:** physical qubits specifically designed to perform entanglements to other communication qubits on non-local registers. They are also entangled to other

qubit types within the quantum computers and are primarily used to mitigate information for distributed quantum operations.

- **Logical qubit:** An error-corrected quantum system whose state lies in a two-dimensional Hilbert space.

8.2.1.2.2 Inquiring supported gates

A *native* gate is a gate that can be executed within a "single pulse interval" using one or several simultaneous pulses. If a gate requires multiple pulses in sequence, then it is called a *compound* gate. If a compiler has full knowledge about which gates are native and which are compound, then it can optimize a compiled program in terms of minimal execution time.

The ISA may also support shortcuts for commonly used (sequences) of native gates as if they where single gate. Those shortcuts are called *primitive* gates. An example may be the well-known Pauli gates that are implemented as (a sequence) of rotations. All primitive gates that are not native are assumed to be compound gates.

The ISA should support at least the following gate inquiry instructions:

- *"Get primitive gates":* Return an identifier list of all gates that are understood by the ISA. This includes both single qubit as well as multi qubit gates. Examples are:
  - Rx(a), Ry(b), Rz(c), X, Y, Z, etc
  - CX, CZ, SWAP, CNOT, sqrtSWAP, etc
- *"Get native gates":* Return an identifier list of a subset of primitive gates that can be executed within a single pulse interval.
- "*Get gate matrix*": Returns for each primitive gate the associated matrix defining the operation.
- "*Get gate duration*": Returns for each primitive gate the operation time.
- "*Get lane size*": Return the number of pulses that can be fired simultaneously to a selected set of qubits. TODO. This instruction may be superfluous. A compiler should prevent such an overload by inquiring "get lane matrix". If that overload occurs, a fault should be raised by the ISA.

TODO: Elaborate on the need for possible instructions, that informs higher layers if a qubit is physical or not. If yes, it may be relevant for higher layers to know if it is a data, measurement, or communication qubit. Such an instruction might be identified as "Get qubit properties".

8.2.1.2.3 Inquiring qubit or register performance

Inquiring performance information is a mechanism to enable higher layers, such as the HAL, to extract information about the maximum circuit depth and other quality parameters. This means the number of time slices that can be executed before the calculation becomes unreliable. Examples are specifying the error of primitive gates, expressing a fidelity for each gate, or simply specifying a

maximum circuit depth. This value is related to coherence time of the implementation and other performance parameters of each connection between qubits. Details about these mechanisms are still to be elaborated, but they should facilitate higher layers, such as the HAL, to identify one or more of the following performance parameters:

- "*Get gate performance*" such as:
    - Coherence times, e.g (T1, T2), where "T1" refers to state decoherence time and "T2" refers to phase decoherence time.
    - Fidelity for certain gates.
    - Qubit pulse error rates.
    - Read out error to indicate how accurate a measurement can be
- "*Get instruction duration*": List of durations for each identified instruction (operations, pulses, read out times, etc.). Note that this is a generalization of the afore mentioned "*get gate duration*" instruction
- "*Get qubit properties*": List of properties of each identified qubit e.g (T1, T2, qubit type, position coordinates, etc.)

8.2.1.3 Execution instructions

Execution instructions are instructions that are meant to change the state of qubits for calculation or measurement purposes, or to prepare for such an instruction.

preparation/initialization level:

- "*Reset calculation*": call a procedure for re-calibration the setup from a set of pre-defined calibration procedures. One should only adjust for small changes in the setup due to drift or other imperfections of the setup, which can be done in a reasonably short period of time. As such it does not refer to a full calibration as described in "initialization instructions".
- "*Reset qubits*": Change the states of all qubits of a selected register into predefined ones, which can be identified as their "|0>" state.
- "*Reset flags*": Set all binary flags to "false" of a selected conditional register.
- "*Set flag X*": Set an individual binary flag "X" of a selected conditional register to a selected value.

pulse level:

Control electronics can fire multiple pulses in parallel, and before they are actually fired by the electronics the ISA must support instructions to prepare this for each qubit individually. However the available hardware limits the number of pulses that can be fires simultaneously. When more qubits are to be controlled than this limit, a switching matrix is to be used to reach all qubits of interest. They can offer so called "lanes" to connect a qubit to a particular pulse generator. To save hardware such a switching matrix may support only a restricted set of lanes, and higher layers such as the HAL should be aware of such limitations.

- "*select qubits":* Prepare for the connection of a selected set of qubits with available lanes, through which pulses will be transported thereafter The actual connection may be delayed until a "wait" or "fire" instruction is handled.
- "*select pulses*": Prepare for each lane of interest a predefined pulse that is to be transported thereafter. The actual firing of pulses may be delayed until a "fire" instruction is handled.
- "*fire pulses*": Fire an ensemble of the selected pulses through the selected lanes for a specified duration, when the selection of lanes and pulses is completed. This instruction will actually execute a specific native gate by changing its state into a specific phase. During this duration, the ISA can prepare for other lanes and pulses, to be used for firing a next ensemble of pules.
- "*wait*": impose a selected amount of delay before a next ensemble of pulses can be fired. During this duration, the ISA can prepare for other lanes and pulses, to be used for firing a next ensemble of pules.

gate level:

An ISA has full knowledge on what (sequence of) lanes and pulses are needed to execute primitive gates. This frees higher layers of the burden to know the exact implementation of lanes and required pulses. As such, gate execution instructions can be regarded as higher in abstraction than pulse execution instructions.

- "*select gates*": Prepare for an ensemble of gates to be executed simultaneously on selected qubits. The actual connection may be delayed until a "wait" or "fire" instruction is handled. This includes the selection of potential conditional gates.
- "*fire gates*": Fire a sequence of simultaneous pulse-delay combinations to the selected qubits in order to execute the selected native gates.

measurement level:

- "*select bases*": prepare for an ensemble of selected qubits the pulse from the "basis library", that are needed to measure the state of these qubits in a specific basis.
- "*fire measurements*": Fire a sequence of simultaneous pulse-delay combinations to measure the selected qubits in a specific basis, detects their response and store the results in the associated conditional bits.

8.2.1.4 Fault handling instructions

These instructions are about classical means to handle all kinds of "classical errors". But to avoid confusion with "quantum errors" the word "fault" is consistently used here to denote those "classical errors".

Each time a fault is raised, the setup should increment the associated fault counter. This enables higher layers, such as the HAL, to read-out these fault counters and to perform proper fault handling.
Each time a fault occurs, the ISA can raise an interrupt to higher layers to inform that a fault has occurred. These interrupts can be masked individually to prevent then being raised.

Examples of such fault counters are:

- Raise exceeding pulse duration time.
- Raise parameter overflow/underflow faults (when phase is out of bounds, index of libraries, lanes or qubits are wrong, etc.).
- Raise pulse timing faults (in the event that pulses are too close together, pulsed at too short times, etc.).
- Raise runtime faults (in the event that a channel is used to pulse a qubit that is not contained in channel lane, etc.).
- Raise memory faults (instruction is too long).
- Raise connection faults (if during an active hybrid session server disconnects).
- Raise adjacency faults, if direct entanglement is requested between two non-adjacent qubits.
- Raise lane faults, if a pulse has to be send to a qubit while the associated lane cannot be build-up.

The ISA should therefore support readout and (re)setting instructions of these fault counters and flags, including:

- "S*et fault mask all*": force for all faults that the ISA will raise an interrupt to higher layers when a fault occurs.
- "S*et fault mask X*": force for fault "X" that the ISA will raise an interrupt to higher layers when such a fault "X" occurs.
- *"Get fault last*": returns the identifier (or index) of the last fault being encountered.
- "*Get fault all*": returns an array with the content of all fault counters and flags.
- "*Get fault X*": returns the content of fault counter or flag "X".
- "*Reset fault all*": Resets all fault counters and flags to zero of false.
- "*Reset fault masks*": Resets all masks to false, so that no fault will raise an interrupt to higher layers.
- "*Reset fault X*": Resets fault counter or flag "X" to zero of false.

8.2.1.5 Quantum error correction instructions

These instructions are about informing higher layers such that they can perform a high level of quantum error correction (QEC) or to correct low level quantum errors directly.

TODO: Elaborate on what QEC instructions are needed to let higher layers handle proper quantum error correction. The following considerations are input for that elaboration:

Quantum Error Correction (QEC) unit:

- The QEC unit can be dedicated hardware (FPGA, CPU, GPU, …) connected to the ISA
- The QEC unit can request from the ISA to do measurements and perform gates. E.g., a stabilizer measurement may involve performing two-qubit gates between data qubits and measurement qubits, performing single-qubit gates on the measurement qubits, and performing a measurement of the measurement qubits.
- The measurements results yield the error syndromes, which are passed to the decoder software running on the QEC unit.
- The QEC unit can perform decoding based on error syndromes, wherein the output of decoding is a set of the most likely errors of the quantum state.
- Based on the set of the most likely errors, the QEC unit identifies correction operations (e.g., Pauli flips).
- The QEC unit can transfer correction operation instructions to the ISA for performing correction operations, potentially via a Pauli frame unit.

ISA requirements related to QEC:

- ISA must talk to QEC unit (potentially via a Pauli frame unit)
- ISA may take instructions for combined (non-native) gates which are frequently used for QEC operations, e.g., stabilizer measurements.